

5 WALTZ

5.1 Overview

The module *PulWALTZ*, which contains the class **WALTZ**, facilitates the use of WALTZ pulse cycles in GAMMA nmr simulations. Class **WALTZ** contains parameters which define how WALTZ cycles is to be implemented and provides functions for building WALTZ based waveforms, composite pulses, and pulse trains.

5.2 Chapter Contents

5.2.1 WALTZ Section Listing

Overview	page 5-73
Constructors and Assignment	page 5-75
Pulse Waveform Functions	page 5-75
Composite Pulse Functions	page 5-75
Pulse Cycle Functions	page 5-75
Description	page 5-82
WALTZ Examples	page 5-88

5.2.2 WALTZ Function Listing

Constructors and Assignment

WALTZ	- Constructor for WALTZ parameters	page 5-75
=	- WALTZ parameters assignment	page 5-75

Access Functions

WF	- Set/Get Pulse Channel	Inherited From Class Pulse	page 5-76
WF_WALTZR	- Set/Get RF-Field Strength	Inherited From Class Pulse	page 5-76
WF_WALTZK	- Set/Get RF-Field Phase	Inherited From Class Pulse	page 5-76
WF_WALTZK	- Set/Get RF-Field Offset	Inherited From Class Pulse	page 5-76

Pulse Waveform Functions

WF	- WALTZ-R waveform	page 5-76
WF_WALTZR	- WALTZ-R waveform	page 5-76
WF_WALTZK	- WALTZ-K waveform	page 5-76
WF_WALTZQ	- WALTZ-Q waveform	page 5-76

Composite Pulse Functions

PCmp	- WALTZ-R composite pulse	page 5-77
PCmpWALTZR	- WALTZ-R composite pulse	page 5-77
PCmpWALTZK	- WALTZ-K composite pulse	page 5-77

PCmpWALTZQ- WALTZ-Q composite pulse page 5-77

Pulse Cycle Functions

CycWALTZ4 - WALTZ-4 pulse cycle page 5-78

CycWALTZ8 - WALTZ-8 pulse cycle page 5-78

CycWALTZ16 - WALTZ-16 pulse cycle page 5-78

Output Functions

printBase - Print WALTZ basics Class Pulse Inherited page 5-78

print - Print WALTZ into an output stream page 5-81

<< - Print WALTZ to standard output page 5-81

5.2.3 WALTZ Figures & Tables Listing

WALTZ-R 3 Step Sequence: Composite 180 Pulse- page 5-82

WALTZ-K 5 Step Sequence- page 5-83

WALTZ-4 Pulse Cycle- page 5-84

WALTZ-8 Pulse Cycle- page 5-85

Reading WALTZ Parameters- page 5-88

5.2.4 WALTZ Examples

Reading WALTZ Parameters page 5-88

WALTZ Decoupling page 5-90

WALTZ-16 Decoupling vs. Field page 5-92

WALTZ Types vs. Decoupling page 5-94

WALTZ Decoupling with Relaxation page 5-96

WALTZ Decoupling Profile page 5-99

5.3 Constructors and Assignment

5.3.1 WALTZ

Usage:

```
#include <PulWALTZ.h>
WALTZ()
WALTZ(double gB1, const String& ch, double ph=0, double off=0);
WALTZ(const WALTZ& WP)
```

Description:

The function *WALTZ* is used to create a WALTZ parameter container.

1. *PulWALTZ()* - Creates an “empty” NULL WALTZ parameter. Can be later filled by an assignment.
2. *PulWALTZ(double gB1, const& ch, double ph, double off)* - Sets up WALTZ for having an rf-field strength of *gB1* Hz on the channel specified by *ch*. WALTZ will be applied with an overall phase of *ph* degrees and an offset of *off* Hz.
3. *PulWALTZ(const PulWALTZ &PWF1)* - Constructs an identical PulWALTZ to the inputPWF1.

Return Value:

WALTZ returns no parameters. It is used strictly to create a WALTZ parameter container.

Examples:

```
PulWALTZ PW; // Empty WALTZ parameters
PulWALTZ PW1(538.9, “13C”); // WALTZ @  $\gamma B_1=538.9$  on  $^{13}\text{C}$  channel
PulWALTZ PW3(PW1); // Another WALTZ identical to PW1
```

See Also: =

5.3.2 =

Usage:

```
#include <PulWALTZ.h>
void WALTZ operator = (PulWALTZ &PWF1)
```

Description:

The unary operator = (the assignment operator) allows for the setting of one WALTZ to another WALTZ. If the WALTZ being assigned to exists it will be overwritten by the assigned WALTZ.

Return Value:

None, the function is void

Example:

```
PulWALTZ PW1(538.9, “13C”); // WALTZ @  $\gamma B_1=538.9$  on  $^{13}\text{C}$  channel
PulWALTZ PW2 = PW1; // Another WALTZ identical to PW1
```

See Also: PulWALTZ

5.4 Pulse Waveform Functions

5.4.1 WF

5.4.2 WF_WALTZR

5.4.3 WF_WALTZK

5.4.4 WF_WALTZQ

Usage:

```
#include <PulWALTZ.h>
PulWaveform WALTZ::WF()
PulWaveform WALTZ::WF_WaltzR()
PulWaveform WALTZ::WF_WaltzK()
PulWaveform WALTZ::WF_WaltzQ()
```

Description:

The functions **WF_WALTZR**, **WF_WALTZK**, and **WF_WALTZQ**, will return a composite pulses for WALTZ-R, WALTZ-K, and WALTZ-Q respectively. The function **WF** is identical to **WF_WALTZR**.

Return Value:

The function returns an composite pulse.

Example:

```
WALTZ WP;                                // Declare WALTZ Parameters
WP.read("filein.pset")                   // Read in WALTZ Parameters
PulWaveform WR = WP.WFp(sys);             // WALTZ-R waveform
WR = WP.PCmpWALTZR(sys);                  // Also WALTZ-R
PulWaveform WK = WP.WF_WALTZK(sys);       // WALTZ-K waveform
PulWaveform WQ= WP.WF_WALTZQ(sys);       // WALTZ-Q waveform
```

See Also: WALTZ-4, WALTZ-8, WALTZ-16

5.5 Composite Pulse Functions

5.5.1 PCmp

5.5.2 PCmpWALTZR

5.5.3 PCmpWALTZK

5.5.4 PCmpWALTZQ

Usage:

```
#include <PulWALTZ.h>
PulComposite WALTZ::PCmp(const spin_system&)
PulComposite WALTZ::PCmpWaltzR(const spin_system&)
PulComposite WALTZ::PCmpWaltzK(const spin_system&)
PulComposite WALTZ::PCmpWaltzQ(const spin_system&)
```

Description:

The functions *PCmpWALTZR*, *PCmpWALTZK*, and *PCmpWALTZQ*, will return a composite pulses for WALTZ-R, WALTZ-K, and WALTZ-Q respectively. The function *PCmp* is identical to *PCmpWALTZR*.

Return Value:

The function returns an composite pulse.

Example:

```
spin_system sys;           // Declare a spin system
sys.read("filein.sys");    // Read in the spin system
WALTZ WP;                  // Declare WALTZ Parameters
WP.read("filein.pset")     // Read in WALTZ Parameters
PulComposite WR = WP.PCmp(sys); // WALTZ-R composite pulse
WR = WP.PCmpWALTZR(sys);    // Also WALTZ-R
PulComposite WK = WP.PCmpWALTZK(sys); // WALTZ-K composite pulse
PulComposite WQ= WP.PCmpWALTZQ(sys); // WALTZ-Q composite pulse
```

See Also: CycWALTZ-4, CycWALTZ-8, CycWALTZ-16

5.6 Pulse Cycle Functions

5.6.1 CycWALTZ4

5.6.2 CycWALTZ8

5.6.3 CycWALTZ16

Usage:

```
#include <PulWALTZ.h>
PulCycle WALTZ::CycWaltz4(const spin_system&)
PulCycle WALTZ::CycWaltz8(const spin_system&)
PulCycle WALTZ::CycWaltz16(const spin_system&)
```

Description:

The functions *CycWALTZ4*, *CycWALTZ8*, and *CycWALTZ16*, will return a pulse cycles for WALTZ-4, WALTZ-8, and WALTZ-16 respectively.

Return Value:

The function returns an pulse cycle.

Example:

```
spin_system sys;                // Declare a spin system
sys.read("filein.sys");         // Read in the spin system
WALTZ WP;                       // Declare WALTZ Parameters
WP.read("filein.pset")          // Read in WALTZ Parameters
PulCycle W4 = WP.CycWALTZ4(sys); // WALTZ-4 pulse cycle
PulCycle W8 = WP.CycWALTZ8(sys); // WALTZ-8 pulse cycle
PulCycle W16 = WP.CycWALTZ16(sys); // WALTZ-16 pulse cycle
```

See Also: PCmpWALTZ-K, PCmpWALTZ-R, PCmpWALTZ-Q

5.7 Input Functions

5.7.1 read

Usage:

```
#include <PulWALTZ.h>
void WALTZ::read(const String& filename, int idx = -1)
void WALTZ::read(ParameterAVLSet& pset, int idx = -1)
```

Description:

The function read will fill a WALTZ parameter with values in either a specified external ASCII file *filename*, or from a specified parameter set *pset*. If a non-negative index is also included as an argument then the parameters that define the returned WALTZ will be assumed indicated with a “(#)” on each where # is the values of *idx*. See the section on this chapter discussing how to specify WALTZ related parameters in an external file.

Return Value:

void.

Example:

```
WALTZ WP;                                // Declare WALTZ Parameters
WP.read(“filein.pset”)                    // Read in WALTZ Parameters
ParameterAVLSet pset;                     // A GAMMA parameter set
pset.read(“another.pset”);                 // Read in a parameter set
WP.read(pset, 3);                          // Set WALTZ from 3rd in pset
```

See Also: ask_read

5.7.2 ask_read

Usage:

```
#include <PulWALTZ.h>
void WALTZ::ask_read(int argc, char* argv[], int argn)
```

Description:

The function read will interactively set thefill a WALTZ parameter with values in either a specified external ASCII file *filename*, or from a specified parameter set *pset*. If a non-negative index is also included as an argument then the parameters that define the returned WALTZ will be assumed indicated with a “(#)” on each where # is the values of *idx*. See the section on this chapter discussing how to specify WALTZ related parameters in an external file.

Return Value:

void.

Example:

```
WALTZ WP;                                // Declare WALTZ Parameters
WP.read("filein.pset")                   // Read in WALTZ Parameters
ParameterAVLSet pset;                     // A GAMMA parameter set
pset.read("another.pset");                 // Read in a parameter set
WP.read(pset, 3);                          // Set WALTZ from 3rd in pset
```

See Also: `ask_read`

5.8 Output Functions

5.8.1 `print`

Usage:

```
#include <PulWALTZ.h>
ostream& WALTZ::print(ostream& ostr)
```

Description:

The function ***print*** will output WALTZ parameters into a specified output stream ***ostr***.

Return Value:

void.

Example:

```
WALTZ WP;                                // Declare WALTZ Parameters
WP.read("filein.pset")                   // Read in WALTZ Parameters
WP.print(cout);                           // Send WALTZ values to screen
```

See Also: <<

5.8.2 <<

Usage:

```
#include <PulWALTZ.h>
ostream& WALTZ::print(ostream& ostr)
```

Description:

The operator << will output WALTZ parameters to standard output.

Return Value:

void.

Example:

```
WALTZ WP;                                // Declare WALTZ Parameters
WP.read("filein.pset")                   // Read in WALTZ Parameters
cout << WP;                              // Send WALTZ values to screen
```

See Also: `print`

5.9 Description

5.9.1 Introduction

The functions in module **PulWALTZ** and Class **WALTZ** (contained in module **PulWALTZ**), is designed to facilitate the use of WALTZ¹ pulse trains in GAMMA NMR simulation programs. In GAMMA, as in an NMR experiment, we should like to use WALTZ pulse trains as individual steps in a general pulse sequence, including use in variable delays as part of multi-dimensional experiments and/or use in pulse trains during acquisition steps.

5.9.2 WALTZ Parameters

A variable of type WALTZ contains only primitive parameters. In particular, it contains the four values which define a pulse waveform: 1.) A **# steps**, 2.) An **rf-field strength**, 3.) An **rf-phase** 4.) An **rf-offset**. These parameters can be used to completely determine how to set up composite pulses such as that used in a WALTZ pulse train.

5.9.3 WALTZ Waveforms & Composite Pulses

The simplest WALTZ sequence is based on a 3 step composite 180 pulse. The pulses are applied with the same rf-strength but vary in their applied length and phase. The details are shown in the following figure, the composite pulse called WALTZ-R.

WALTZ-R 3 Step Sequence: Composite 180 Pulse

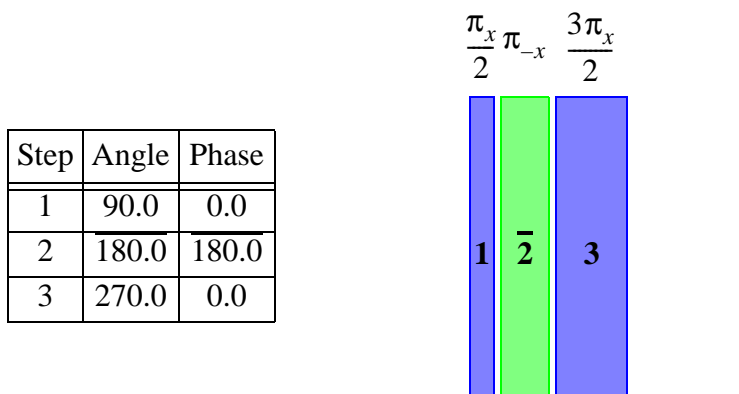


Figure 7-1 The basic WALTZ 3-step waveform. The blue steps indicate pulses that are applied without any phase shift whereas the green step indicates a pulse with a 180 phase shift (as reflected in the table). Shorthand notation is used for this sequence, 123, where each integer reflects multiples of a 90 pulse and bar indicates the phase shift.

The WALTZ-R composite pulse is used to build WALTZ-4 pulse cycles. Additional composite pulses are used in other WALTZ sequences. The next is WALTZ-K, a five step composite pulse as shown in the next figure.

1.

WALTZ-K 5 Step Sequence

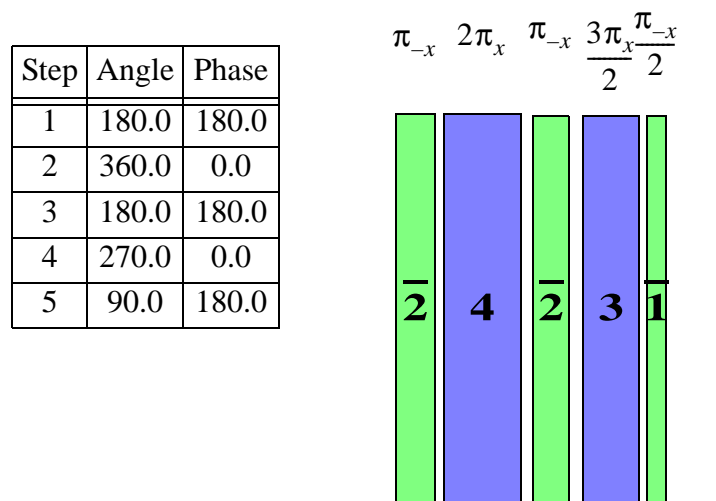


Figure 7-1 The basic WALTZ-K 5-step waveform. The blue steps indicate pulses that are applied without any phase shift whereas the green step indicates a pulse with a 180 phase shift (as reflected in the table). Shorthand notation is used for this sequence, $\bar{2}4\bar{2}3\bar{1}$, where each integer reflects multiples of a 90 pulse and bar indicates the phase shift.

WALTZ-Q composites are used in making WALTZ-16 pulse cycles.

WALTZ-Q 9 Step Sequence

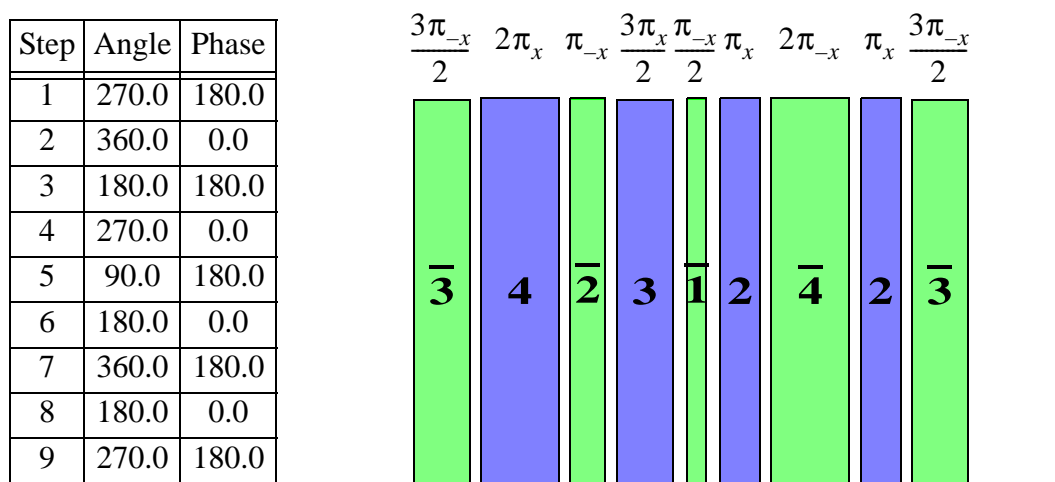


Figure 7-1 The basic WALTZ-Q 9-step waveform. The blue steps indicate pulses that are applied without any phase shift whereas the green step indicates a pulse with a 180 phase shift (as reflected in the table). Shorthand notation is used for this sequence, $\bar{3}4\bar{2}3\bar{1}2\bar{4}2\bar{3}$, where each integer reflects multiples of a 90 pulse and bar indicates the phase shift.

5.9.4 WALTZ Pulse Cycles

The simplest WALTZ based pulse cycle is WALTZ-4. This sequence cycles the basic composite 180 sequence, WALTZ-R, through a 4-step phase change. Using “R” to represent the WALTZ-R base composite pulse, the base WALTZ-4 cycle sequence is given by $RR\bar{R}\bar{R}$ where \bar{R} implies a 180 phase shifted R. This relationship is shown in the figure below.

WALTZ-4 Pulse Cycle

Step	Angle	Phase	Type	Cycle
1	90.0	0.0	1	R
2	180.0	180.0	$\bar{2}$	
3	270.0	0.0	3	
4	90.0	0.0	1	R
5	180.0	180.0	$\bar{2}$	
6	270.0	0.0	3	
7	90.0	180.0	$\bar{1}$	\bar{R}
8	180.0	0.0	2	
9	270.0	180.0	$\bar{3}$	
10	90.0	180.0	$\bar{1}$	\bar{R}
11	180.0	0.0	2	
12	270.0	180.0	$\bar{3}$	

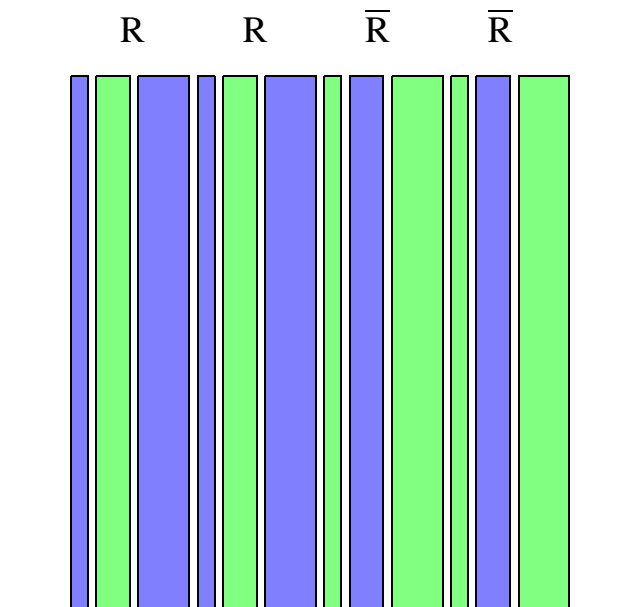


Figure 7-2 The WALTZ-4 pulse cycle. Four WALTZ-R composite pulses are linked together with the last two having their phase angle shifted by 180 degrees. Using R to designate the WALTZ-R and a bar to indicate a 180 phase shift, WALTZ-4 can be described as repeating $RR\bar{R}\bar{R}$. The blue steps above indicate pulses that are applied without any phase shift whereas the green step indicates a pulse with a 180 phase shift (as reflected in the table).

There are two important properties of periodic decoupling sequences in an ideal situation. The first is that their performance is unaffected by phase inversion. The second is that they are unaffected by a cyclic permutation of some part of the sequence. However, in actuality there are small residual effects from choosing one possibility over the other. To compensate, one simply couples sequences together with varying phase and permutation.

The WALTZ-8 sequence cycles the 5 step composite WALTZ-K through a 4-step phase change. Using “K” to represent the WALTZ-K composite pulse, the WALTZ-8 cycle sequence is given by $KKKK$ where \bar{K} implies a 180 phase shifted K. This relationship is shown in the figure below.

WALTZ-8 Pulse Cycle

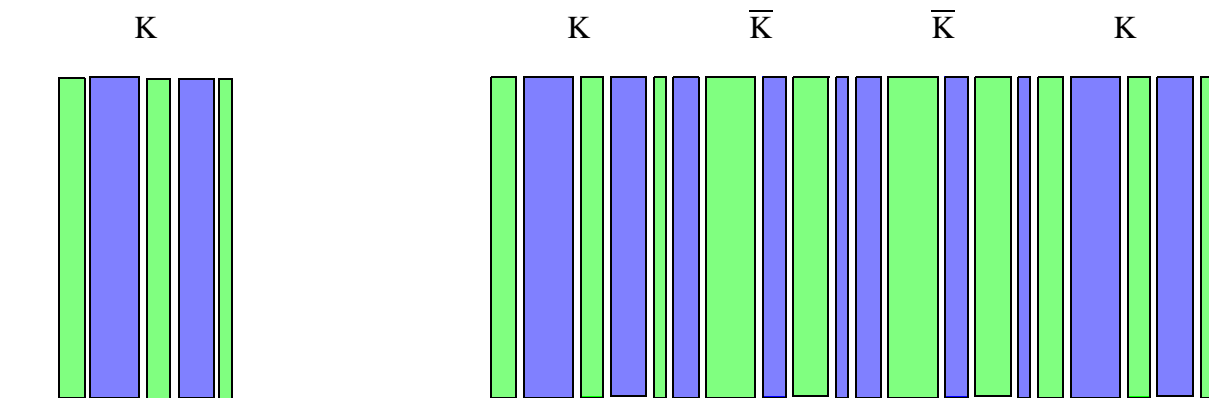


Figure 7-1 The basic 5-step WALTZ-K waveform (on the left) is repeated 4 times to produces the WALTZ-8 cycle. The blue steps indicate pulses that are applied without any phase shift whereas the green step indicates a pulse with a 180 phase shift (as reflected in the table). Shorthand notation is used for this sequence, KKKK. These four steps are repeated as long as WALTZ-8 is to be applied.

The WALTZ-16 sequence cycles the 9 step composite WALTZ-Q through a 4-step phase change. Using “Q” to represent the WALTZ-Q composite pulse, the WALTZ-16 cycle sequence is given by QQQQ where \bar{Q} implies a 180 phase shifted Q. This relationship is shown in the figure below.

WALTZ-16 Pulse Cycle

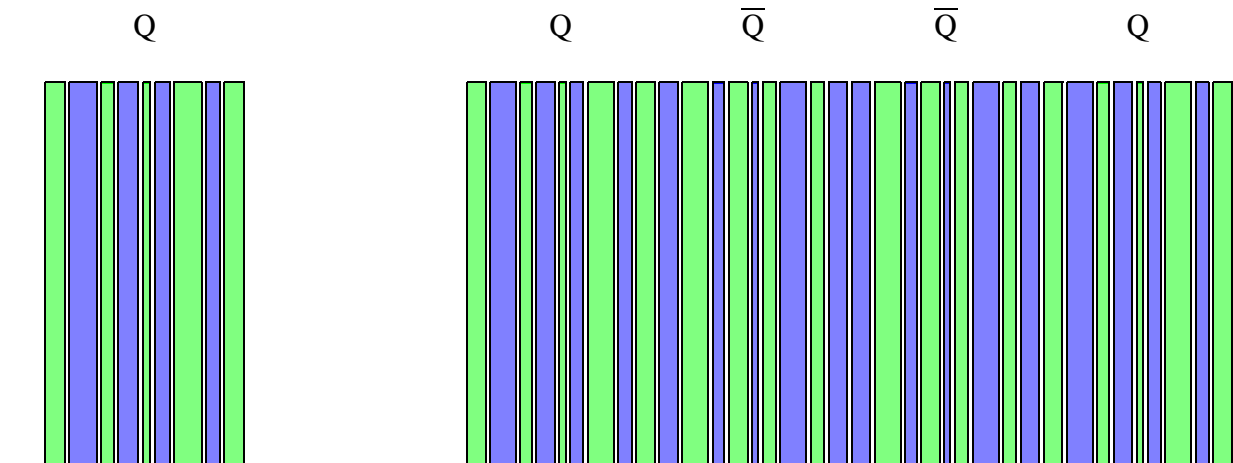


Figure 7-1 The basic 9-step WALTZ-Q waveform (on the left) is repeated 4 times to produces the WALTZ-16 cycle. The blue steps indicate pulses that are applied without any phase shift whereas the green step indicates a pulse with a 180 phase shift (as reflected in the table). Shorthand notation is used for this sequence, QQQQ. These four steps are repeated as long as WALTZ-16 is to be applied.

5.9.5 WALTZ Parameters

This section describes how an ASCII file may be constructed that is self readable by a WALTZ variable. The file can be created with an editor of the users choosing and is read with the WALTZ member function “read”. This provides for an extremely flexible and program independent means of implementing WALTZ in NMR simulations.

The WALTZ (ASCII) input file is scanned for the specific parameters which specify the pulse-delay parameters¹: rf-strength, rf-phase, rf-offset, and the rf-channel. These parameters are recognized by certain keywords, as shown in the following table.

Table 1: Spin System Parameters

Parameter Keyword	Assumed Units	Examples Parameter (Type) : Value - Statement
WALTZgamB1	Hz	WALTZgamB1 (1) : 600.0 - Field Strength (Hz)
WALTZiso	none	WALTZiso (2) : 19F - WALTZ rf pulse channel
WALTZphi	degrees	WALTZphi (1) : 2.0 - WALTZ rf phase (deg)

The order in which these parameters reside in the ASCII file is of no consequence.

The format of each parameter is quite simple and general for all GAMMA parameters. At the beginning of a line the parameter keyword is written followed by an optional index number in parenthesis. This is then followed by one or more blanks and then an integer in parentheses. The integer corresponds to the type of parameter value: 0 = integer, 1 = floating point, or 2 = string. Following the parenthesis should be at least one blank then a colon to indicate the parameter value follows. The parameter value is then written followed by some blanks then a hyphen followed by an optional comment.

There is one major restriction; keywords and string parameters cannot contain blanks. For example, v (0) is unknown, v(0) is. The string value 19 F is unknown, 19F is fine. If multiple WALTZ pulse-delay steps need to be defined in the same file then simply put an index on all parameters associated with a desired WALTZ and read the parameters using that index.

To read the file, see the documentation for function read (or ask-read) and look at the example programs in this chapter along with their input files.

Channel: WALTZiso

This parameter is optional. It will define which isotope channel the WALTZ rf-pulse will be applied on. If no channel is specified GAMMA will assume that all spins in the system being treated are affected by the rf. Thus if no channel is specified and WALTZ is utilized in an NMR simulation the system should be homonuclear or the same WALTZ should be desired on all channels (same offset, phase, etc.)

1. Note that the ASCII file must contain viable parameters in GAMMA format. Indeed, the file is a GAMMA parameter set and, as such, may contain any amount of additional information along with the valid WALTZ parameters.

Channel: WALTZphi

This parameter is optional. It will define the rf-phase of the WALTZ pulse. If no phase is specified it will be taken to be zero.

Pulse Strength: WALTZgamB1

The parameters { WALTZang, WALTZgamB1, WALTZtp } work together. WALTZgamB1 se the pulse strength if either WALTZang or WALTZgamB1 have also been specified. If only WALTZgamB1 is specified amongst the three an error will result when reading these parameters to define WALTZ. If all three parameters have been specified then WALTZgamB1 will be ignored, the strength set by { WALTZang, WALTZtp }

5.10 WALTZ Examples

5.10.1 Reading WALTZ Parameters

To keep GAMMA programs using WALTZ sequences versatile, users will want to keep all WALTZ specifications undetermined in the code. As the program runs, WALTZ settings are either specified interactively and/or read in from an external ASCII (parameter) file. This section gives examples of the latter case. The figure below shows an ASCII parameter file on the left and some GAMMA program code on the right.

Reading WALTZ Parameters

WALTZphi	(1) : 90.0	- WALTZ overall rf-phase (deg)	WALTZ WP;	// Declare WALTZ parameters
WALTZiso	(2) : 1H	- WALTZ pulse channel	WP.read("WALTZ.pset");	// Read WALTZ from file
WALTZgamB1	(1) : 983.0	- WALTZ pulse strength (Hz)	WP.read("WALTZ.pset", 3);	// Read WALTZ from file
WALTZiso(3)	(2) : 19F	- WALTZ pulse channel	WP.ask_read(argc, argv, 1);	// Read WALTZ from file
WALTZgamB1(3)(2)	: 2045.9	- WALTZ pulse strength (Hz)	WP.ask_read(argc, argv, 2, 3);	// Read WALTZ from file

Figure 7-2 Typical WALTZ parameters (left) and the GAMMA code which reads them. The parameters are contained in an ASCII file which the code reads to set up use of WALTZ.

The ASCII parameter file (on the left) is taken to be called “WALTZ.pset” and is read in by the program code. Thus one can change the WALTZ parameters independent of the GAMMA code. The ASCII file format is typical of GAMMA parameter sets: The line ordering is of no consequence, the column spacing is not important, the end “- comments” can be left off, and additional lines of text or parameters may be included.

The GAMMA code is color coded with the parameters they read in the previous figure. Thus the second line (blue) will read the blue parameters and set up WALTZ with a strength of 983 Hz on the proton channel with an overall phase of 90 degrees. Similarly, the next line (green) will read the parameters colored green from the same ASCII file but sets up WALTZ with a strength of 2.0459 kHz on the 19F channel (no phase, no offset).

The next line will interactively ask the user to supply a filename where the program can get to WALTZ parameters. This filename (in this case “WALTZ.pset”) will be prompted for unless the user specifies the file on the command line when the program is executed. The following line does the same but reads the WALTZ parameters indexed with a “3” from the file.

Using a combination of these commands, the user has complete flexibility in defining one or more WALTZ sequences in the same GAMMA program. The WALTZ parameters can be easily changed by either changing their values in the ASCII file and/or changing the filename given to the program. See the section of WALTZ parameters to see which parameters can be used in setting up WALTZ sequences. See the other programs in this chapter for full examples GAMMA programs using them.

5.10.2 WALTZ Decoupling

In this section we shall produce a simple 1D NMR spectrum under WALTZ-16 decoupling. A hard 90 pulse will be applied to a chosen spin system on the acquisition channel. Then WALTZ-16 will be applied on the decoupler channel during acquisition. The resulting FID will be apodized and Fourier transformed, the NMR spectrum put on screen using Gnuplot. Note that relaxation and exchange effects will be ignored in this simulation. The code for simple WALTZ-16 decoupling is shown below:

```
/* WALTZdec0.cc *****
**
**
**          GAMMA Decoupling Example Program
**
** This program uses the class WALTZ to perform a simple decoupling
** simulation. A hard ideal pulse will be applied to an input spin
** system. Subsequently, an acquisition will be performed with WALTZ-16
** decoupling applied on a specified channel.
**
** Author:   S.A. Smith
** Date:    3/9/98
** Update:  3/9/98
** Version: 3.5.4
** Copyright: S. Smith. Modify this program as you see fit for personal
** use, but you must leave the program intact if redistributed
**
*****
**/
```

```
#include <gamma.h>
#
main(int argc, char* argv[])
{
    cout << "\n\t\t\tWALTZ Decoupling\n\n";
    int qn = 1;
    spin_system sys;
    WALTZ WP;
    String filename = sys.ask_read(argc,argv,qn++);
    cout << sys;
    WP.read(filename);
    PulCycle PCyc = WP.CycWALTZ16(sys);
    String IsoD = sys.symbol(0);
    if(sys.heteronuclear())
        query_parameter(argc, argv, qn++,
```

```
// Query index
// Declare a spin system
// WALTZ parameters
// Ask for/read in the system
// Have a look (for setting SW)
// Read in WALTZ parameters
// Construct WALTZ-16 cycle
// Detection/pulse channel
// If heteronuclear system
// ask for detection channel
```

```
        "\n\tDetection Isotope? ", IsoD);
double SW;
query_parameter(argc, argv, qn++,
        "\n\tSpectral Width (Hz)? ", SW);
int npts = 1024;
query_parameter(argc, argv, qn++,
        "\n\tBlock Size? ", npts);
double lwvh = 1.0;
query_parameter(argc, argv, qn++,
        "\n\tApodization (Hz)? ", lwvh);
double td = 1/SW;
double R = (lwvh/2)*HZ2RAD;
double tt = (npts-1)*td;
gen_op H = Ho(sys);
gen_op Det = Fm(sys, IsoD);
gen_op sigma0 = sigma_eq(sys);
gen_op sigmap = lypuls(sys,sigma0,IsoD, 90.);
row_vector data = PCyc.FID(npts,td,Det,sigmap);
row_vector exp = Exponential(npts,tt,0.0,R,0);
row_vector fidap = product(data,exp);
data = FFT(fidap);
GP_1D("spec.asc", data, 0, -SW/2, SW/2);
GP_1Dplot("spec.gnu", "spec.asc");
}
```

```
// Spectral width
// Get desired spectral width
```

```
// Block size (must be base 2)
// Get block size
```

```
// Half-height linewidth
// Ask for apodization strength
```

```
// Set dwell time
// Set apodization rate
// Total FID length
// Set isotropic Hamiltonian
// Set detection operator to F-
// Set density mx equilibrium
// This is 1st 90 pulse
// Perform acquisition under WALTZ-16
// Here is an exponential
// Apodized FID
// Transformed FID -> spectrum
// Output ASCII file
// Plot to screen using Gnuplot
```

The first half of this program simply sets the parameters up interactively. Note that both the spin system and the WALTZ parameters are contained in the same file who's name the user must specify.

The second half of the program does the simulation. The FID is acquired using the pulse cycle function "FID" and the pulse cycle has been set to WALTZ-16. The last few lines apodize and transform the FID then spit the plot out on the screen.

Addition of relaxation & exchange effects and/or changing the 1st pulse to non-ideal will require only minor modifications of a few lines.

The input parameter (ASCII) file is listed on the following page. It contains parameters for both the spin system and the WALTZ settings.

Example of a WALTZ decoupling input file (WALTZdec.sys)

SysName	(2) : WALTZ	- Name of the Spin System
NSpins	(0) : 4	- Number of Spins in the System
Iso(0)	(2) : 1H	- Spin Isotope Type
Iso(1)	(2) : 1H	- Spin Isotope Type
Iso(2)	(2) : 1H	- Spin Isotope Type
Iso(3)	(2) : 13C	- Spin Isotope Type
v(0)	(1) : 105.0	- Chemical Shifts in Hz
v(1)	(1) : -174.32	- Chemical Shifts in Hz
v(2)	(1) : 15.0	- Chemical Shifts in Hz
v(3)	(1) : 0.0	- Chemical Shifts in Hz
J(0,1)	(1) : 10.0	- Coupling Constants in Hz
J(0,2)	(1) : 7.9	- Coupling Constants in Hz
J(0,3)	(1) : 22.0	- Coupling Constants in Hz
J(1,2)	(1) : 2.8	- Coupling Constants in Hz
J(1,3)	(1) : 32.0	- Coupling Constants in Hz
J(2,3)	(1) : 18.3	- Coupling Constants in Hz
Omega	(1) : 400	- Spect. Freq. in MHz (1H based)
WALTZphi	(1) : 0	- WALTZ pulse phase (deg)
WALTZiso	(2) : 13C	- WALTZ pulse channel
WALTZgamB1	(1) : 1500.0	- WALTZ pulse strength (Hz)

When the program (WALTZdec0.cc) is compiled its execution will produce a plot on screen if the Gnuplot program is available. Assuming the executable is called a.out, the following command will produce a spectrum:

a.out WALTZdec.sys 1H 500 1024 .5

The command “a.out” alone will prompt you for input values. Had you input the above command (or parameters) the spectrum should appear as shown in the following figure.

¹³C Decoupled Spectrum Using WALTZ-16

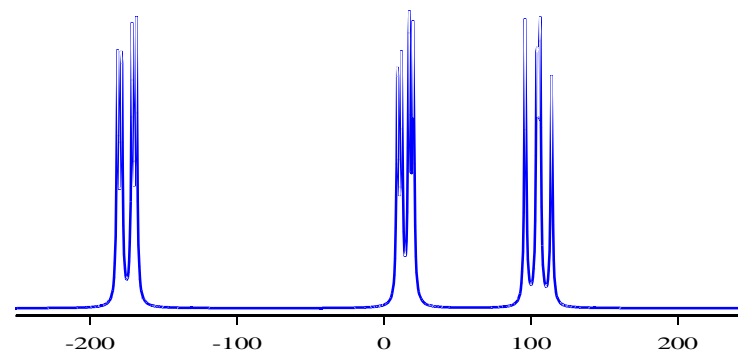


Figure 7-3 The spectrum produced using the program WALTZdec0.cc with input parameter file WALTZdec.sys. The decoupler was applied to the 13C channel with a 1.5 kHz field strength. Detection was on the proton channel. 1K data points were collected using a spectral width of 500 Hz. The data was processed with a 0.5 Hz line-broadening.

By either editing the input file or specifying a different input file, the spectrum can be radically altered. For example, by setting the WALTZ pulse strength to zero one obtains the following spectrum.

¹³C Coupled Spectrum, Zero Strength WALTZ-16

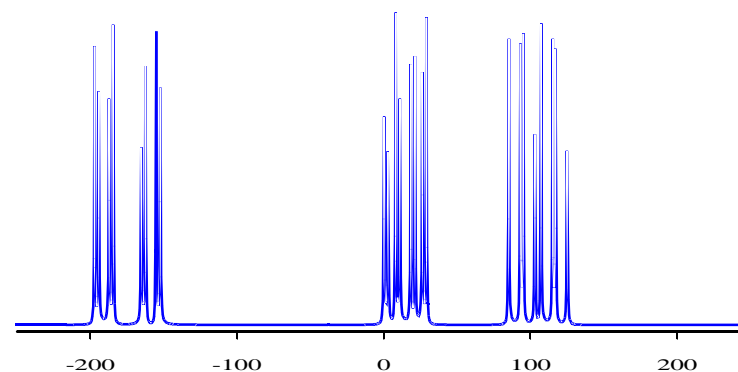


Figure 7-4 Same as previous figure but with zero decoupler field.

5.10.3 WALTZ-16 Decoupling vs. Field

We can readily modify the previous program to loop over differing rf-field strengths and determine how well WALTZ-16 does at decoupling. In this case we will just read in a series of gB1 values from an external ASCII file and loop over them producing a 1D spectrum at each value. We'll spit out all the spectra in a single stack plot.

```

/* WALTZdecstk1.cc *****
**
**          GAMMA Decoupling Test Program
**
** This program uses the class WALTZ to perform a simple decoupling
** simulation. A hard ideal pulse will be applied to an input spin
** system. Subsequently, an acquisition is taken with WALTZ-16
** decoupling applied on a specified channel. This pulse-acquisition
** process will be repeated over a series of specified decoupler
** rf-field strengths. The decoupled spectra will be output on screen
** if Gnuplot is available. The stack plot is also output in
** FrameMaker MIF format.
**
** Author:   S.A. Smith
** Date:    3/11/98
** Update:  3/11/98
** Version: 3.5.4
** Copyright: S. Smith. You can modify this program as you see fit
**           for personal use, but you must leave the program intact
**           if you re-distribute it.
**
*****/

#include <gamma.h>

main(int argc, char* argv[])
{
    cout << "\n\t\t\tWALTZ Decoupling Vs. Decoupler Strength\n\n";
    int qn = 1; // Query index
    spin_system sys; // Declare a spin system
    WALTZ WP; // WALTZ parameters
    String filename; // Input filename
    filename = sys.ask_read(argc,argv,qn++); // Ask for/read in the system
    cout << sys; // Have a look (for setting SW)
    WP.read(filename); // Read in WALTZ parameters
    cout << WP;
    PulCycle PCyc;

```

```

    query_parameter(argc, argv, qn++,
        "\n\tFile of Field Strengths? ", filename); // Ask for field strength file
    int N; // Number of field strengths
    double* gB1s = GetDoubles(filename, N); // Get array of field strengths

    String IsoD = sys.symbol(0); // Detection/pulse channel
    if(sys.heteronuclear()) // If heteronuclear system
        query_parameter(argc, argv, qn++, // ask for detection channel
            "\n\tDetection Isotope? ", IsoD);
    double SW; // Spectral width
    query_parameter(argc, argv, qn++, // Get desired spectral width
        "\n\tSpectral Width (Hz)? ", SW);
    int npts = 1024; // Block size (must be base 2)
    query_parameter(argc, argv, qn++, // Get block size
        "\n\tBlock Size? ", npts);
    double lwhh = 3.0; // Half-height linewidth
    query_parameter(argc, argv, qn++, // Ask for apodization strength
        "\n\tApodization (Hz)? ", lwhh);
    double td = 1/SW; // Set dwell time
    double R = (lwhh/2)*HZ2RAD; // Set apodization rate
    double tt = (npts-1)*td; // Total FID length
    gen_op H = Ho(sys); // Set isotropic Hamiltonian
    gen_op Det = Fm(sys, IsoD); // Set detection operator to F-
    gen_op sigma0 = sigma_eq(sys); // Set density mx equilibrium
    gen_op sigmap = lypuls(sys,sigma0,IsoD, 90.); // This is 1st 90 pulse
    row_vector data, exp, fidap;
    matrix datamx(N,npts);
    for(int i=0; i<N; i++)
    {
        WP.strength(gB1s[i]); // Set WALTZ field strength
        PCyc = WP.CycWALTZ16(sys); // Set WALTZ-16 pulse cycle
        data = PCyc.FID(npts,td,Det,sigmap); // Perform acquisition
        exp = Exponential(npts,tt,0.0,R,0); // Here is an exponential
        fidap = product(data,exp); // Apodized FID
        data = FFT(fidap); // Transformed FID -> spectrum
        datamx.put_block(i,0,data);
    }
    double Nm1 = double(N-1);
    String AF("stk.asc");
    String GF("stk.gnu");
    GP_stack(AF, datamx, 0,1,N,0.0,Nm1);
    GP_stackplot(GF, AF);
    FM_stack("stk.mif", datamx, 1.5, 1.5, 1);
}

```

The modifications from the previous program are obvious. An external ASCII file is used to specify a list of decoupler field strengths and these are read into the program. These fields are looped over, a new spectrum computed at each decoupler strength.

The spectra are put into a matrix which is given to the Gnuplot routines for display as a stack plot on screen. In addition, the stack plot is output in FrameMaker MIF format for incorporation into documents in an editable form. The latter is shown in the following figure.

¹³C WALTZ-16 Decoupling Versus RF-Field Strength

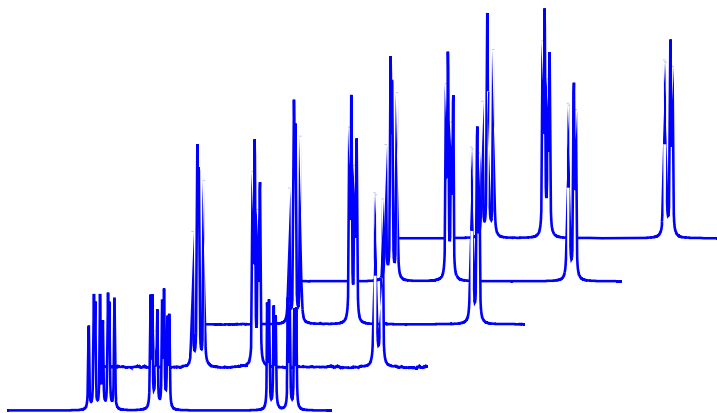


Figure 7-5 Proton spectra produced using the program WALTZdec1.cc with input parameter file WALTZdec.sys and decoupler strength file WALTZdecBs. The decoupler was applied to the ¹³C channel with field strengths shown. Detection was on the proton channel. 1K data points were collected using a spectral width of 500 Hz. The data was processed with a 1.5 Hz line-broadening. Note: the baseline “noise” exists because of my use of an asynchronous acquisition. By setting a spectral width that is commensurate with the WALTZ-16 cycle length that will disappear (see function FIDsync, its used in the next example).

When the program (WALTZdecstk1.cc) is compiled its execution will produce a stack plot on screen if the Gnuplot program is available. Assuming the executable is called a.out, the following command will produce the plot shown in the previous figure:

a.out WALTZdec.sys WALTZdecBs 1H 500 1024 1.5

The ASCII file WALTZdecBs contains a list of rf-field strengths (in Hz) that the program used. The file has a single field strength per line and is shown next.

WALTZ decoupling rf-field input file (WALTZdecBs)

```
0
200
400
600
800
```

Unlike GAMMA parameter set files (such as WALTZdec.sys) this file is simple ASCII and cannot have anything other than a single floating point or integer value per line. No additional comments may be included.

5.10.4 WALTZ Types vs. Decoupling

For something different, let's compare how well the different WALTZ sequences perform. The base WALTZ composite 180 pulse, WALTZ-R, is cycled to produce WALTZ-4. The base composite pulse is altered to produce WALTZ-8 and WALTZ-16, which are based on WALTZ-K and WALTZ-Q composite pulses respectively. The variations are improvements which are meant to suppress artifacts that can result from pulse mis-calibration.

In this example we will generate WALTZ decoupled spectra using all six of these sequences. Since we are using “perfect” rectangular pulses and neglecting relaxation, the resulting spectra should be (nearly) identical. Note the difference in the use of Composite Pulses (WALTZ-R, WALTZ-K, WALTZ-Q) relative to Pulse Cycles (WALTZ-4, WALTZ-8, and WALTZ-16). Their associated functionality in GAMMA is very similar although the cycles are automatically accounting for the phase changes in the composite pulses. Also, note that more obvious differences between the sequences should arise when relaxation effects are included and/or we intentionally mis-set the pulse lengths.

¹³C WALTZ Decoupling Versus WALTZ Type

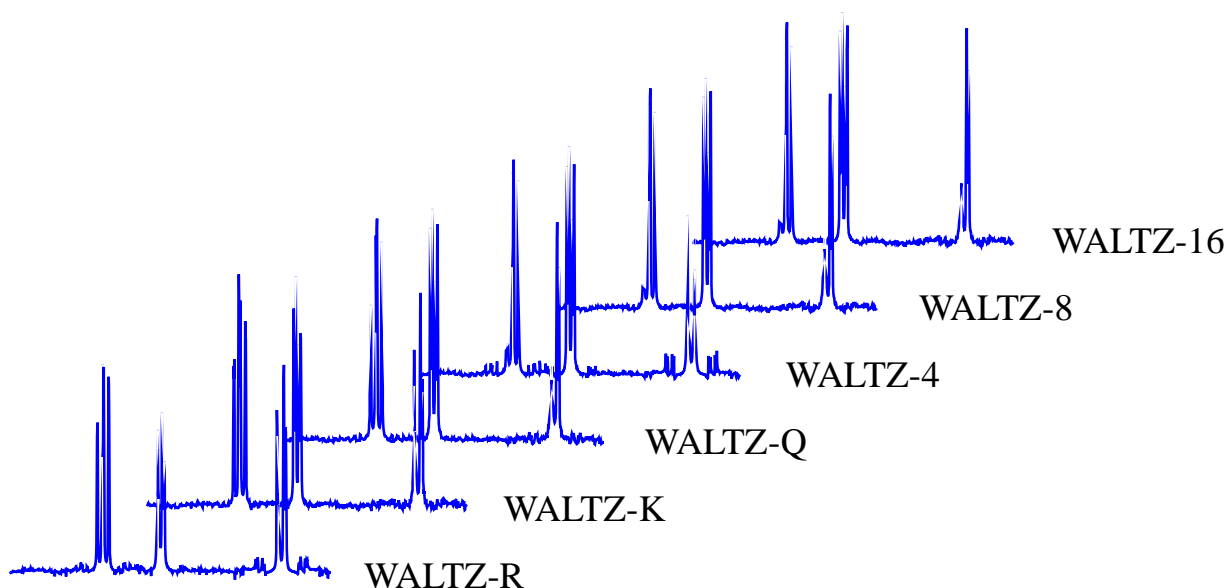


Figure 7-6 Proton spectra produced using the program WALTZtypes1.cc with input parameter file WALTZ-dec.sys. The decoupler was applied to the ¹³C channel and detection was on the proton channel. 1K data points were collected using a spectral width of 500 Hz. The data was processed with a 1.0 Hz line-broadening. Note: the baseline “noise” exists because of my use of an asynchronous acquisition. By setting a spectral width that is commensurate with the various WALTZ sequence lengths that will disappear (see function FIDsync).

The program below produced the previous stack plot.

```

/* WALTZtypes1.cc *****
**
**                                     GAMMA Decoupling Test Program
**
** This program uses the class WALTZ to perform a simple decoupling
** simulation. A hard ideal pulse will be applied to an input spin
** system. Subsequently, an acquisition is taken with one of the
** WALTZ decoupling sequences applied on a specified channel. This
** pulse-acquisition process will be repeated over a series of WALTZ
** sequences (WALTZ-{R,K,Q,4,8,16}). The decoupled spectra will be
** output on screen if Gnuplot is available. The stack plot is also
** output in FrameMaker MIF format.
**
** Author:   S.A. Smith
** Date:     3/11/98
** Version:  3.5.4
**
*****/

#include <gamma.h>

main(int argc, char* argv[])
{
    cout << "\n\t\t\t\tWALTZ Decoupling Vs. WALTZ Type\n\n";
    int qn = 1;                                // Query index
    spin_system sys;                            // Declare a spin system
    WALTZ WP;                                  // WALTZ parameters
    String filename;                            // Input filename
    filename = sys.ask_read(argc,argv,qn++);    // Ask for/read in the system
    cout << sys;                                // Have a look (for setting SW)
    WP.read(filename);                          // Read in WALTZ parameters
    cout << WP;

    String IsoD = sys.symbol(0);                // Detection/pulse channel
    if(sys.heteronuclear())                     // If heteronuclear system
        query_parameter(argc, argv, qn++,     // ask for detection channel
            "\n\tDetection Isotope? ", IsoD);
    double SW;                                 // Spectral width
    query_parameter(argc, argv, qn++,          // Get desired spectral width
        "\n\tSpectral Width (Hz)? ", SW);
    int npts = 1024;                           // Block size (must be base 2)
    query_parameter(argc, argv, qn++,          // Get block size
        "\n\tBlock Size? ", npts);
    double lwhh = 3.0;                          // Half-height linewidth
    query_parameter(argc, argv, qn++,          // Ask for apodization strength
        "\n\tApodization (Hz)? ", lwhh);
    double td = 1/SW;                           // Set dwell time
    double R = (lwhh/2)*HZ2RAD;                 // Set apodization rate

    double tt = (npts-1)*td;
    gen_op H = Ho(sys);
    gen_op Det = Fm(sys, IsoD);
    gen_op sigma0 = sigma_eq(sys);
    gen_op sigmap = lypuls(sys,sigma0,IsoD, 90.); // Total FID length
    row_vector data, exp, fidap;                // Set isotropic Hamiltonian
                                              // Set detection operator to F-
                                              // Set density mx equilibrium
                                              // This is 1st 90 pulse
    int N = 3;                                  // Number of WALTZ cycle types
    int M = 3;                                  // Number of WALTZ composite types
    matrix datamx(N+M,npts);
    PulComposite Comps[M];
    Comps[0] = WP.PCmpWALTZR(sys);              // Set WALTZ-R composite pulse
    Comps[1] = WP.PCmpWALTZK(sys);              // Set WALTZ-K composite pulse
    Comps[2] = WP.PCmpWALTZQ(sys);              // Set WALTZ-Q composite pulse
    PulCycle Cycles[N];
    Cycles[0] = WP.CycWALTZ4(sys);              // Set WALTZ-4 pulse cycle
    Cycles[1] = WP.CycWALTZ8(sys);              // Set WALTZ-8 pulse cycle
    Cycles[2] = WP.CycWALTZ16(sys);             // Set WALTZ-16 pulse cycle
    PulComposite PComp;
    PulCycle PCyc;
    int i;
    for(i=0; i<M; i++)
    {
        PComp = Comps[i];
        data = PComp.FID(npts,td,Det,sigmap); // Perform acquisition
        exp = Exponential(npts,tt,0.0,R,0);    // Here is an exponential
        fidap = product(data,exp);              // Apodized FID
        data = FFT(fidap);                     // Transformed FID -> spectrum
        datamx.put_block(i,0,data);
    }
    for(int j=0; j<N+M; j++)
    {
        PCyc = Cycles[j];
        data = PCyc.FID(npts,td,Det,sigmap);    // Perform acquisition
        exp = Exponential(npts,tt,0.0,R,0);    // Here is an exponential
        fidap = product(data,exp);              // Apodized FID
        data = FFT(fidap);                     // Transformed FID -> spectrum
        datamx.put_block(j,0,data);
    }
    double Nm1 = double(N+M-1);
    String AF("stk.asc");
    String GF("stk.gnu");
    GP_stack(AF, datamx, 0,1,N+M,0.0,Nm1);
    GP_stackplot(GF, AF);
    FM_stack("stk.mif", datamx, 1.5, 1.5, 1);
}

```

5.10.5 WALTZ Decoupling with Relaxation

How can we include the effects of relaxation (and/or exchange) when we decouple? Since we already have a couple of programs that simulate decoupled spectra under WALTZ without relaxation, we need only make the proper modifications to them and their input files in order to obtain the simulation(s) we want.

Lets review a few of basic changes we'll need. First, rather than working with an isotropic spin system (`spin_system`) in our program, we need to work with a oriented spin system that is moving isotropically. That is, a spin system that keeps track of dipolar, CSA, and quadrupolar tensors for all spins or spin-pairs. Thus we need to replace *spin_system* with *sys_dynamic*. Second, when the system is read in from an external ASCII file it will look for tensor quantities as well as dynamical values (correlation times). Next we will have to create a relaxation matrix and Liouvillian that defines how the system evolves. And lastly, we'll have to use an FID function that includes that evolves under the defined Liouvillian so that relaxation (and exchange) are accounted for.

This seems like it is complicated, but in fact amounts only to about few lines of code changes. In this example, I'll modify the previous program to include relaxation.... but I'll remove the loop over the different WALTZ types and just allow the user to choose one. A couple of results are shown in the next figure and we'll see the code after that.

¹³C WALTZ Decoupling Under Relaxation

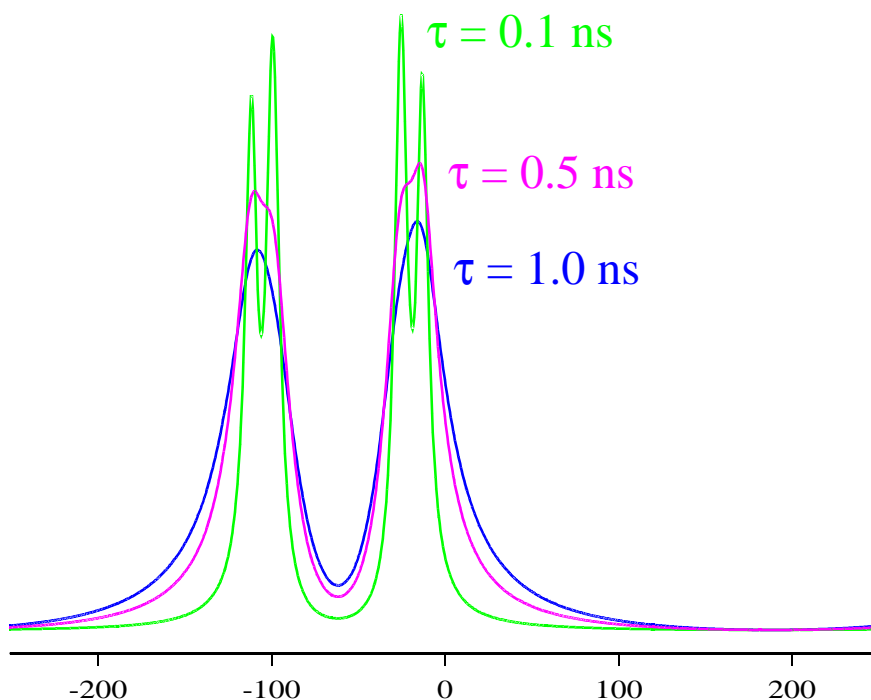


Figure 7-7 Proton spectra produced using the program WALTZdec2.cc with input parameter file WALTZdec3.dsys. The decoupler was applied to the ¹³C channel and detection was on the proton channel. 1K data points were collected using a spectral width of 500 Hz. The data was processed without line-broadening. The input file is given on a subsequent page, WALTZdec3.dsys. Successive plots were made with the same input file except for the correlation times (kept spherical) altered as reflected on the plot.

```

/* WALTZdec2.cc *****
**
**          GAMMA Decoupling Test Program
**
** This program uses the class WALTZ to perform a simple decoupling
** simulation. A hard ideal pulse will be applied to an input spin
** system. Subsequently, an acquisition is performed with WALTZ
** decoupling applied on a specified channel. The decoupled spectrum
** will be output on screen if Gnuplot is available. The spectrum
** is also output in FrameMaker.MIF format.
**
** Note: This program is identical to WALTZdec1.cc except that it
** includes the effects of relaxation.
**
** Author:   S.A. Smith
** Date:     3/30/98
** Update:   3/30/98
** Version:  3.5.4
** Copyright: S. Smith. You can modify this program as you see fit
**            for personal use, but you must leave the program intact
**            if you re-distribute it.
**
*****/

#include <gamma.h>
main(int argc, char* argv[])
{
    cout << "\n\t\t\tWALTZ Decoupling With Relaxation\n\n";
    int qn = 1; // Query index
    sys_dynamic sys; // Declare a spin system
    WALTZ WP; // WALTZ parameters
    String filename; // Input filename
    filename = sys.ask_read(argc,argv,qn++); // Ask for/read in the system
    cout << sys; // Have a look (for setting SW)
    WBRExch WBR; // Relaxation parameters
    WBR.read(filename, sys); // Read in relaxation parameters
    cout << WBR; // Have a look at relaxation settings
    WP.read(filename); // Read in WALTZ parameters

    cout << "\n\tWALTZ Decoupling Schemes:\n"; // Ask fo WALTZ type
    cout << "\n\t\tRepeated WALTZ-R (1)";
    cout << "\n\t\tRepeated WALTZ-K (2)";
    cout << "\n\t\tRepeated WALTZ-Q (3)";
    cout << "\n\t\tWALTZ-4 Cycle (4)";
    cout << "\n\t\tWALTZ-8 Cycle (5)";
    cout << "\n\t\tWALTZ-16 Cycle (6)";
    int wt;
    query_parameter(argc, argv, qn++, // Get number of steps
        "\n\t\tWALTZ Type? ", wt);
    if(wt<1 || wt>6) wt=1; // Insure [1,6]

    PulComposite PCmp;
    PulCycle PCyc;
    switch(wt)
    {
        case 1: PCmp = WP.PCmpWALTZR(sys); break;
        case 2: PCmp = WP.PCmpWALTZK(sys); break;
        case 3: PCmp = WP.PCmpWALTZQ(sys); break;
        case 4: PCyc = WP.CycWALTZ4(sys); break;
        case 5: PCyc = WP.CycWALTZ8(sys); break;
        case 6: PCyc = WP.CycWALTZ16(sys); break;
    }
    String IsoD = sys.symbol(0); // Detection/pulse channel
    if(sys.heteronuclear()) // If heteronuclear system
        query_parameter(argc, argv, qn++, // ask for detection channel
            "\n\tDetection Isotope? ", IsoD);
    double SW; // Spectral width
    query_parameter(argc, argv, qn++, // Get desired spectral width
        "\n\tSpectral Width (Hz)? ", SW);
    int npts = 1024; // Block size (must be base 2)
    query_parameter(argc, argv, qn++, // Get block size
        "\n\tBlock Size? ", npts);
    double lwhh = 3.0; // Half-height linewidth
    query_parameter(argc, argv, qn++, // Ask for apodization strength
        "\n\tApodization (Hz)? ", lwhh);
    double td = 1/SW; // Set dwell time
    double R = (lwhh/2)*HZ2RAD; // Set apodization rate
    double tt = (npts-1)*td; // Total FID length
    gen_op H = Ho(sys); // Set isotropic Hamiltonian
    gen_op Det = Fm(sys, IsoD); // Set detection operator to F-
    gen_op sigma0 = sigma_eq(sys); // Set density mx equilibrium
    gen_op sigmap = lypuls(sys,sigma0,IsoD, 90.); // This is 1st 90 pulse
    super_op L = WBR.REX(sys,H); // Get relaxation superoperator
    row_vector data;
    if(wt<=3)
    {
        PCmp.setRelax(sys,L); // Put relaxation into pulse
        data=PCmp.FIDR(npts,td,Det,sigmap,1); // Perform acquisition
    }
    else
    {
        PCyc.setRelax(sys,L); // Put relaxation into pulse
        data=PCyc.FIDR(npts,td,Det,sigmap,1); // Perform acquisition
    }
    row_vector exp=Exponential(npts,tt,0.0,R,0); // Here is an exponential
    row_vector fidap = product(data,exp); // Apodized FID
    data = FFT(fidap); // Transformed FID -> spectrum
    GP_1D("spec.asc", data, 0, -SW/2, SW/2); // Output ASCII file
    GP_1Dplot("spec.gnu", "spec.asc"); // Plot to screen using Gnuplot
    FM_1D("spec.mif", data,14,14,-SW/2, SW/2); // Plot in FrameMaker MIF
}

```

The previous simulation was given the input file “WALTZdec3.dsys” which is shown below. This file contains a spin system, WALTZ parameters, and relaxation parameters.

SysName	(2) : WALTZ	- Name of the Spin System
NSpins	(0) : 3	- Number of Spins in the System
Iso(0)	(2) : 1H	- Spin Isotope Type
Iso(1)	(2) : 1H	- Spin Isotope Type
Iso(2)	(2) : 13C	- Spin Isotope Type
v(0)	(1) : 105.0	- Chemical Shifts in Hz
v(1)	(1) : 20.0	- Chemical Shifts in Hz
v(2)	(1) : 0.0	- Chemical Shifts in Hz
J(0,1)	(1) : 12.0	- Coupling Constants in Hz
J(0,2)	(1) : 22.0	- Coupling Constants in Hz
J(1,2)	1) : 28.0	- Coupling Constants in Hz
Coord(0)	(3) : (0.0, 0.0, 0.0)	- Coordinate Point (A)
Coord(1)	(3) : (0.0, 0.0, 1.1)	- Coordinate Point (A)
Coord(2)	(3) : (0.0, 0.0, -1.1)	- Coordinate Point (A)
Taus	(3) : (0.5, 0.5, 0.5)	- Correlation Times (ns)
Omega	(1) : 400	- Spec. Freq. in MHz (1H based)

WALTZphi	(1) : 0	- WALTZ pulse phase (deg)
WALTZiso	(2) : 13C	- WALTZ pulse channel
WALTZgamB1	(1) : 3000.0	- WALTZ pulse strength (Hz)

Rlevel	(0) : 4	- Relaxation Computation Level
Rtype	(0) : 0	- Relaxation Computation Type
RDD	(0) : 1	- Dipolar Relaxation Flag
RDDdfs	(0) : 0	- Dipolar DFS Flag
RCC	(0) : 0	- CSA Relaxation Flag
RCCdfs	(0) : 0	- CSA DFS Flag
RQQ	(0) : 0	- Quad Relaxation Flag
RQQdfs	(0) : 0	- Quad DFS Flag

RDQ	(0) : 0	- Dip-Quad Relaxation Flag
RDQdfs	(0) : 0	- Quad DFS Flag
RDC	(0) : 0	- Dip-CSA Relaxation Flag
RDCdfs	(0) : 0	- Dip-CSA DFS Flag
RQC	(0) : 0	- Quad-CSA Relaxation Flag
RQCdfs	(0) : 0	- Quad-CSA DFS Flag

For successive simulations (as shown in the previous figure) the input correlation times were altered. The three lines used were

Taus	(3) : (0.1, 0.1, 0.1)	- Correlation Times (ns)
Taus	(3) : (0.5, 0.5, 0.5)	- Correlation Times (ns)
Taus	(3) : (1.0, 1.0, 1.0)	- Correlation Times (ns)

In all three cases, WALTZ-16 was used. Each spectrum size was set to 1024 points and no line-broadening was used in the processing. The spectral width was set to 500 Hz and detection was on the proton channel.

5.10.6 WALTZ Decoupling Profile

In this section we shall attempt to produce a WALTZ decoupling profile. A hard 90 pulse will be applied to a simple heteronuclear spin system on the acquisition channel. Then WALTZ decoupling will be applied on the decoupler channel during acquisition. The user will specify which WALTZ sequence to use. The resulting FID will be apodized and Fourier transformed. This pulse-delay process will be repeated for differing offsets on the decoupler channel. Each spectrum will be plotted with its center at the offset frequency to produce the profile.

The really no significant differences between this and our previous calculations. To determine a profile one typically uses the simplest spin system (here a two spin heteronuclear system). The 1D spectrum is recalculated after either moving the decoupler rf offset or, equivalently, moving all decoupler isotope channel chemical shifts. The spectra are all just put into a single vector, offset so their respective centers are set to be referenced to the decoupler offset value.

Here are some of the results from the GAMMA program given on the following page.

WALTZ Decoupling Profiles

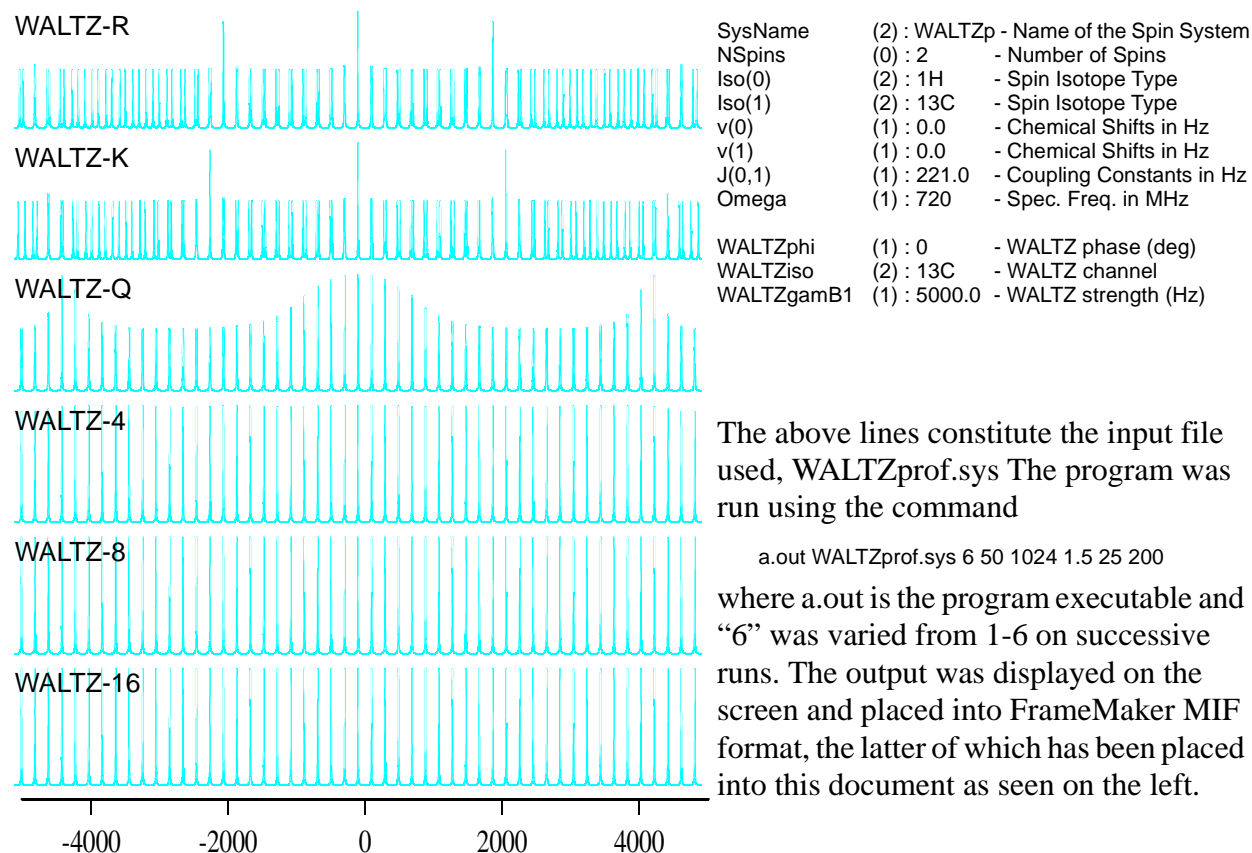


Figure 7-8 WALTZ decoupling profiles produced from the program WALTZprof2.cc. Decoupling was performed on the carbon channel in a ^{13}C - ^1H two spin system. The decoupling rf-field strength was set to 5 kHz and the scalar coupling to 221 Hz. A linebroadening of 1.5 Hz was used in processing the spectra. The block size was 1K and the offset increment set to 200 Hz.

```

/* WALTZprof2.cc *****
**
**                                     GAMMA Decoupling Test Program
**
** This program uses the class WALTZ to perform a simple decoupling
** simulation. A hard ideal pulse will be applied to a simple two
** spin heteronuclear system. Subsequently, an acquisition will be
** performed with WALTZ decoupling applied on one the channel which
** is not being detected. This process will be repeated over a range
** of decoupler offsets. The result is a WALTZ decoupler profile.
** The profile will be plotted on screen if Gnuplot is available on
** the system. The profile will also be output in FrameMaker MIF.
**
** Author:   S.A. Smith
** Date:     4/7/98
** Update:   4/7/98
** Version:  3.5.4
**
*****/

#include <gamma.h>

main(int argc, char* argv[])
{
    cout << "\n\t\t\tWALTZ Decoupling Profile\n\n";
    // Read In Spin System & WALTZ Parameters
    int qn = 1; // Query index
    spin_system sys; // Declare a spin system
    WALTZ WP; // WALTZ parameters
    String filename; // Input filename
    filename = sys.ask_read(argc,argv,qn++); // Ask for/read in the system
    cout << sys; // Have a look (for setting SW)
    if(sys.spins()!=2 || sys.homonuclear())
        cout << "\n\tWarning! This program has been"
        << " set up for a two spin heteronuclear"
        << " system. Results on other systems"
        << " can be unpredictable.....";
    WP.read(filename); // Read in WALTZ parameters
    // Set WALTZ Decoupling Scheme
    cout << "\n\tWALTZ Decoupling Schemes:\n"; // Ask fo WALTZ type
    cout << "\n\t\tRepeated WALTZ-R (1)";
    cout << "\n\t\tRepeated WALTZ-K (2)";
    cout << "\n\t\tRepeated WALTZ-Q (3)";
    cout << "\n\t\tWALTZ-4 Cycle (4)";
    cout << "\n\t\tWALTZ-8 Cycle (5)";
    cout << "\n\t\tWALTZ-16 Cycle (6)";
    int wt;
    query_parameter(argc, argv, qn++, // Get number of steps
        "\n\t\tWALTZ Type? ", wt);

    if(wt<1 || wt>6) wt=1; // Insure [1,6]

    PulComposite PCmp;
    PulCycle PCyc;
    switch(wt)
    {
        case 1: PCmp = WP.PCmpWALTZR(sys); break;
        case 2: PCmp = WP.PCmpWALTZK(sys); break;
        case 3: PCmp = WP.PCmpWALTZQ(sys); break;
        case 4: PCyc = WP.CycWALTZ4(sys); break;
        case 5: PCyc = WP.CycWALTZ8(sys); break;
        case 6: PCyc = WP.CycWALTZ16(sys); break;
    }

    // Set Acquisition and Profile Parameters
    String IsoD = sys.symbol(0); // Detection/pulse channel
    String IsoG = WP.channel(); // Decoupler channel
    if(IsoD == IsoG) // Try and set channel to
        IsoD = sys.symbol(1); // not be the decoupling one
    double SW; // Spectral width
    query_parameter(argc, argv, qn++, // Get desired spectral width
        "\n\tSpectral Width (Hz)? ", SW);
    int npts = 1024; // Block size (must be base 2)
    query_parameter(argc, argv, qn++, // Get block size
        "\n\tBlock Size? ", npts);
    double lwhh = 3.0; // Half-height linewidth
    query_parameter(argc, argv, qn++, // Ask for apodization strength
        "\n\tApodization (Hz)? ", lwhh);
    int NO = 30; // # Of Offsets (on each side)
    query_parameter(argc, argv, qn++, // Get # offsets
        "\n\tNumber of Positive Decoupler Offsets? ", NO);
    double offset; // Get # offsets
    query_parameter(argc, argv, qn++, // Get # offsets
        "\n\tDecoupler Offset Per Step (Hz)? ", offset);

    // Set Up Variables Consistent Through All Offsets
    double R = (lwhh/2)*HZ2RAD; // Set apodization rate
    gen_op Det = Fm(sys, IsoD); // Set detection operator to F-
    gen_op sigma0 = sigma_eq(sys); // Set density mx equilibrium
    gen_op sigma = lypuls(sys,sigma0,IsoD, 90.); // This is 90 detection pulse
    row_vector data(npts); // Block for acquisition

    // Set Up Variables Global Over Full Profile
    row_vector profile((2*NO+1)*npts, complex0); // Block for profile
    double totaloff = double(NO)*offset; // Total offset at end
    row_vector fidap; // Block for apodized FID
    if(wt <=3) SW = PCmp.FIDsync(SW); // Synchronize dwell times
    else SW = PCyc.FIDsync(SW);
    cout << "\n\tSynch Spectral Width " << SW;
    double td = 1/SW; // Set dwell time
    double tt = (npts-1)*td; // Total FID length

```

```

row_vector exp=Exponential(npts,tt,0.0,R,0);    // Block for apodization
//      Loop Over Offsets, Calculate Profile
int K=0;                                         // Point index in profile
sys.offsetShifts(-NO*offset, IsoG);            // Set 1st profile offset
cout << "\n\tProfile Offset ";
for(int ov=-NO; ov<=NO; ov++)                  // Loop over offsets
{
    printIdx(cout, ov);                        // Output offset index
    switch(wt)
    {
        case 1: PCmp = WP.PCmpWALTZR(sys); break;
        case 2: PCmp = WP.PCmpWALTZK(sys); break;
        case 3: PCmp = WP.PCmpWALTZQ(sys); break;
        case 4: PCyc = WP.CycWALTZ4(sys); break;
        case 5: PCyc = WP.CycWALTZ8(sys); break;
        case 6: PCyc = WP.CycWALTZ16(sys); break;
    }
    if(wt<=3) data=PCmp.FIDR(npts,td,Det,sigmap); // Perform acquisition
    else data=PCyc.FID(npts,td,Det,sigmap);
    fidap = product(data,exp);                  // Apodized FID this offset
    data = FFT(fidap);                          // Spectrum this offset
    profile.put_block(0, K, data);              // Put spectrum in profile
    sys.offsetShifts(offset, IsoG);             // Move system to next offset
    K += npts;                                  // Adjust profile point index
}

double F = totaloff + SW/2;                    // Final plot frequency
GP_1D("prof.asc", profile, 0, -F, F);          // Output profile ASCII data
GP_1Dplot("prof.gnu", "prof.asc");            // Plot to screen using Gnuplot
FM_1D("prof.mif", profile, 14,14,-F, F);      // Plot in FrameMaker MIF
}

```

dled in an efficient manner in simulations. In principle composite pulses could simply be pulse cycles with only 1 phase. Similarly, pulse cycles could be composite pulse where the number of steps would be multiplied by the number of phase changes the waveform is put through during the cycle.

Both of these will likely be less efficient in calculating decoupling effects! This is because GAMMA internally reuses Hamiltonians and propagators when possible. However, I will likely add in the ability to generate a WALTZ-K pulse cycle and a WALTZ-16 composite pulse since it will simplify some GAMMA programs (at the expense of computation efficiency).

It is evident that use of the WALTZ pulse cycles is superior to use of repeated composite pulses without the compensating phase cycle (i.e. WALTZ- $\{4,8,16\}$ works better than WALTZ- $\{R,K,Q\}$). It also is clear that, at least for this two spin system without relaxation effects, the WALTZ-16 is the best of these WALTZ sequences.

Unfortunately the source code for this example was well over a page long, mostly due to allowing the user to choose between decoupling schemes. Part of the length is also due to GAMMA use of Composite Pulses (WALTZ- $\{R,K,Q\}$) versus Pulse Cycles (WALTZ- $\{4,8,16\}$).

This difference is set in GAMMA so that the two types are internally han-